System Management CSCI
Redundancy Management CSC
Thor DP3 Part 1

December 18, 1997
Thor DP3.0

Table of Contents

Table of Figures

1. Redundancy Management CSC

1.1 Redundancy Management CSC Introduction

1.1.1 Redundancy Management CSC Overview

Release Notes:

- **The Redundancy Management CSC is being delivered in two drops for Thor. The first drop is scheduled for mid-January and consists of executing SCT APIs for a local computer, and stub SSI APIs. The second drop contains full Thor functionality. This document focuses on the interface definition (APIs) to ensure that other CSCIs can meet their deadlines. Internal design and documentation of that design will be discussed at a second Thor DP3.**

The Redundancy Management CSC monitors, and maintains the health of the RTPS. It does this by monitoring the health of both the software and hardware in the system. *If failures are detected and a recovery mechanism is in place, Redundancy Management implements the recovery*. All failures cause generation of a System Message.

Redundancy Management also manages the RTPS Test Set. The configuration of this Test Set is reflected in the System Configuration Table (SCT) during operations. This table specifies the hardware and software configuration, both logical and physical. The SCT specifies the resources allocated to Test Sets, which then support specific Activities. The static portion, and initial values for much of the dynamic part of the SCT are generated off-line and loaded by Redundancy Management into memory at initialization. Redundancy Management maintains the dynamic portion of the table and makes all data available to displays and other applications.

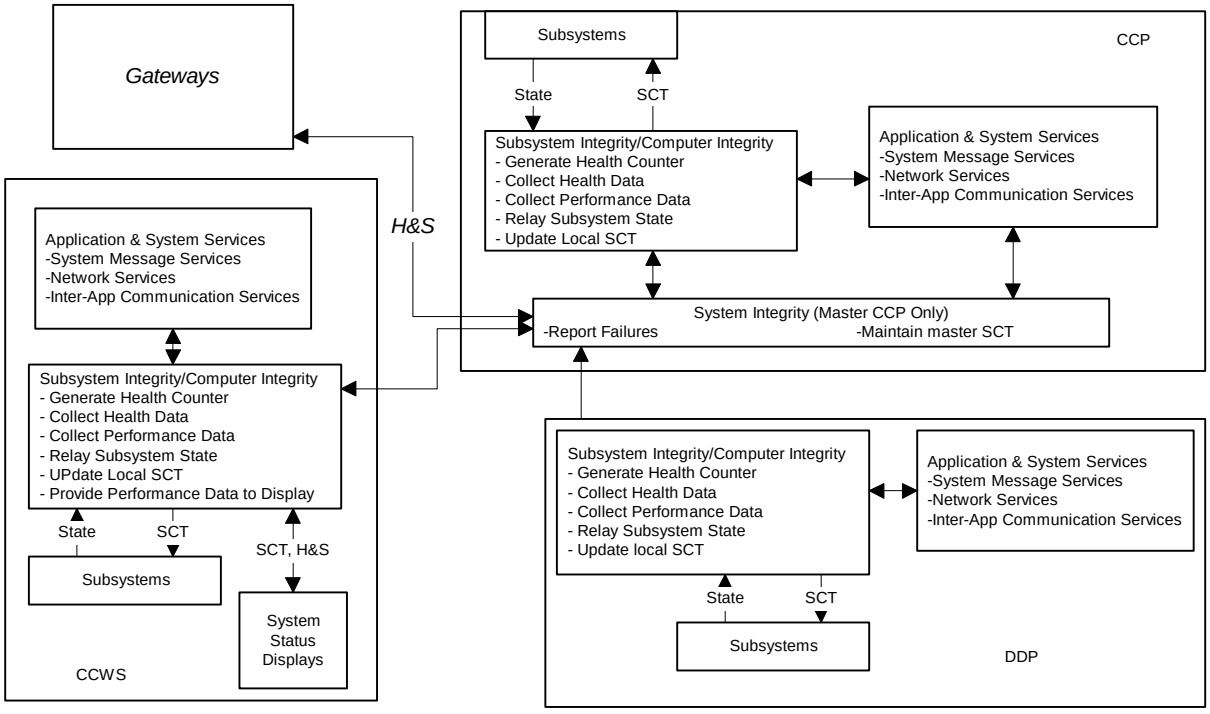


Figure 1. Redundancy Management Conceptual Data Flow Diagram

1.1.2 Redundancy Management CSC Operational Description

As shown in Figure 2, Redundancy Management is composed of five parts: *Set Integrity*, System Integrity, Subsystem Integrity, Computer Integrity and System Configuration Table. Each Integrity monitors the integrity of its parts and reports the results of this analysis to a higher level. The collected integrity is reflected in the SCT.

Parts of the Redundancy Management CSC execute in the DDPs, CCPs, CCWSs, and the Ops/CM Server. Subsystem and Computer Integrity equivalents execute in the Gateways, but are not part of this CSC. *Set Integrity executes in the Set Master CCWS*. The SCT is available to any computer that executes any part of the Redundancy Management CSC.

Computer Integrity executes in each Computer, monitors the health and status of the computer, and records standard hardware performance data. It makes the health data available to any local Subsystem Integrity operation. Both the health and performance data are provided to System Integrity.

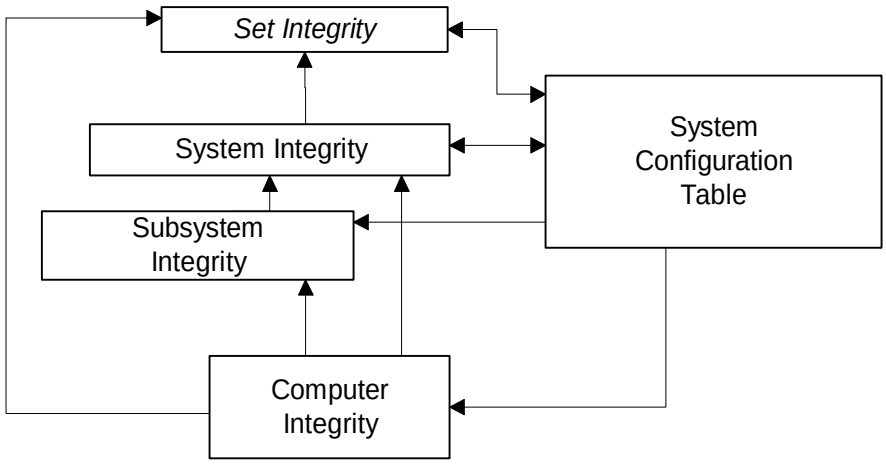


Figure 2. Logical Redundancy Management Organization

Subsystem Integrity executes in each Subsystem and uses the Computer health and information provided from applications in the Subsystem to generate Subsystem Health and the Health Counter FD. This health and status report is provided to System Integrity. Using a combination of the Health and Status report and information from other subsystems, System Integrity makes a determination on the health of the Subsystem. Any changes in the health and status are recorded in the System Configuration Table, which is viewable on the System Status Display.

Based on collected data, System Integrity determines the health of each subsystem and keeps the SCT current. It also provides general health of the Test Set to Set Integrity. System Integrity executes in the Set Master CCP.

1.2 Redundancy Management CSC Specifications

1.2.1 Redundancy Management CSC Groundrules

- Definitions
    - An Activity is a named operation performed as part of a test. It may consist of lower level Activities.
    - Computer is defined to be a physical box that contains one or more CPUs or Processors
        - Computers communicate with other Computers over the Network
        - Any peripherals in the same physical box such as a disk drive, memory, cards are also considered to be part of the Computer.
    - CPU and Processor are used interchangeably to refer to actual compute engine of a computer. Many modern Computers have more than one CPU.
    - Role: Active or Standby. In some cases Hot Spare may be considered a role, and certain configurations may have other unique roles.
    - State: (In Configuration, Loaded, Communicating, Go, In ORT)
    - Group: a collection of computers that cannot be logically separated, and are therefore assigned to single test set. This is also known as a Control Group
- Redundant subsystems are actually composed of two Subsystems, one designated as active, one designated as standby. In this document, Subsystem refers to the Active or Standby Subsystem, not the combined pair.
- Executable code with an API is provided for the CCPs, DDPs, CCWSs, and Ops/CM Server. This replaces the Application Services provided API identified in the System Integrity thread.
- Dependencies:
    - *Ideally, System Integrity will reuse Data Fusion and Constraint Management software to build failure data from health and status information.*
    - Interface for generation of System Messages (System Services)
    - OPS/CM will ensure that the correct SCT data set is downloaded to the computer prior to initializing SCT software.
    - *ITS Software must register processes with Subsystem Integrity as they are initiated and terminated.*
    - *Each system process must generate a heartbeat*
    - *Each user application process must generate either an heartbeat, or an Application Health Counter FD.*
- External Interfaces:
    - Gateways will supply FDs as required by System Integrity.
    - Redundancy Management will generate the following System Status FDs:
        - SCCnnHC CCPnn Health Counter
        - SCCnnPRmmR CCPnn, Process MM is Running
        - SCCnnPRmmF CCPnn, Process MM failed
        - SCCssSTAT CCP Subsystem ss is Communicating
        - SDDnnHC DDPnn Health Counter
        - SDDnnPRmmR DDPnn, Process MM is Running
        - SDDnnPRmmF DDPnn, Process MM failed
        - SDDssSTAT DDP Subsystem ss is Communicating
        - SWnnnHC CCWSnnn Health Counter

- SWnnnPRmmR CCWSnnn Process mm is Running
- SWnnnPRmmF CCWSnnn Process mm failed
- SWSnnnSTAT CCWSnnn is Communicating

– System Messages
  - Missed Heartbeat
  - Computer State Change
  - Subsystem State Change

- Thor will be delivered in two parts, an informal drop with group 2 (scheduled for 1/20), with a full drop in Group 8 (4/20).
  – Group 2 Content:
    - Stub APIs for SSI Registration, Heartbeat and Error Reporting
      - The intent is to provide APIs that allow the uses to correctly code their software. These APIs return no data, so stub implementations do not need any functionality underneath.
    - Locally operating APIs for the System Configuration Table.
      - The interfaces will execute and return data from the local copy of the SCT. The local copy will be static unless updated on the local machine through APIs. Any changes made to the local copy through the API will affect the SCT on that processor only.
    - An offline tool to generate the System Configuration Table
      - This will be a Microsoft Access Database. Table editors will be available as will an export routine to make the information available to the processors.
  – Group 8 Content (Full Thor Content):
    - The SCT is maintained across the system.
    - SSI Interfaces function, with data relayed to SI
    - SI reports failures, but takes no action in response to failures
    - Improved Access interface

1.2.2 Redundancy Management CSC Functional Requirements

The functional requirements for System Integrity are arranged in the following major functions:

1 Set Integrity
2 System Integrity
3 Subsystem Integrity
4 Computer Integrity
5 System Configuration Table

1 Set Integrity

Set Integrity provides the equivalent of System Integrity, but for all Test Sets in the system and any computers not currently assigned to a Test Set

2 System Integrity

System Integrity evaluates the operations on computers and subsystems. It provides data for display at the System Status Display, and reports any errors or failures as System Messages. *When automatic recovery such as switchover is possible, it determines when to recover and directs the recovery*.

2.1 System Integrity will be a Redundant Subsystem

2.2 There will be one System Integrity *Redundant Pair* in each Test Set.

2.3 System Integrity will execute in the Test Set Master CCP.

2.4 If the received Subsystem Health Counter contains the next expected value and current State of the Computer is Go, no change will be made to the SCT. Note that the next expected value will typically be 1 greater than the current value, but it may be a rollover.

2.5 If the received Subsystem Health Counter contains a value other than the next expected value, a Missed Health Counter System Message will be generated.

2.6 If the received Subsystem Health Counter contains a value other than the next expected value, the Computer on which the subsystem executes will be placed in a No Go state.

2.7 If an expected Health Counter is not received, the Computer on which the Subsystem executes will be placed in a No Go State. Note that the missing Health Counters must be detected without receiving a packet from the computer.

2.8 If a Computer is placed in a No Go State, all Subsystems on the Computer will be placed in a No Go State.

2.9 If a Computer is placed in a No Go State, a Computer State Change System Message will be generated.

2.10 Upon receipt of a Subsystem Health Message, System Integrity will update the SCT to reflect any changes in the state of the subsystem.

2.11 If a Subsystem is placed in a No Go State, a Subsystem State Change System Message will be generated.

2.12 System Integrity will use data other than the Subsystem Health Message and Subsystem Health Counter to determine the Subsystem Health.

2.13 If an Active Redundant Subsystem is placed in a No Go State, System Integrity will trigger the Standby Subsystem to transit to the Active Role.

2.14 If a subsystem is put in No Go state, the subsystem will be commanded to terminate.

3 Subsystem Integrity

Subsystem Integrity monitors the health and status of the hardware and software that composes a subsystem. As processes come up, they register with Subsystem Integrity and begin supplying a periodic heartbeat. Subsystem Integrity monitors the heartbeat to ensure that each registered application is still cycling. Subsystem Integrity also provides a mechanism for processes to report errors. Based on the status processes as determined by the heartbeat and any error reports, Subsystem Integrity makes a determination on the health of the subsystem. The primary health indicator used to report the subsystem health is the Subsystem Health Counter. As long as the subsystem is healthy, the counter is periodically incremented and published as an FD. If the subsystem is determined to be unhealthy, the Health Counter is published, but not incremented.

3.1 Subsystem Integrity will provide an API to allow processes to **register** the process and specify its minimum rate.

3.2 Subsystem Integrity will automatically **register** processes that issue an Application Health Counter FD.

3.3 When a process **registers,** Subsystem Integrity will assign the process a CLCS Process Number and generate a "Process is Running" FD containing the mapping between the Process name and the CLCS Process number.

3.4 Subsystem Integrity will provide a Heartbeat API to allow the those processes that registered through the API to check in periodically.

3.5 If a registered Process fails to check in at its specified minimum rate, System Integrity will generate a Process Failed FD. This check in can be either through the API or through continued Application Health Counter FDs.

3.6 A subsystem is considered **healthy** under the following conditions:

3.6.1 Each registered process is generating a heartbeat, and

3.6.2 All critical processes have registered, and

3.6.3 No Fatal Errors have been reported by the processes in the subsystem, and
3.6.4 *TBD*

3.7 Subsystem Integrity will periodically issue a Health Counter FD.

3.8 The first Health Counter FD will be issued when Subsystem Integrity has finished its own initialization.

3.9 If the Subsystem is **healthy**, Subsystem Integrity will increment the Health Counter FD prior to issuing it. Note that this increment may result in a rollover.

3.10 If the Subsystem is not **healthy**, Subsystem Integrity will not increment the Health Counter FD prior to issuing it.

3.11 Multiple Subsystems on a single computer will be supported. The intent of this requirement is to allow for a combined CCP/DDP in an IDE.

3.12 Subsystem Integrity will provide an API that allows a process to record errors detected by the process.

3.13 All errors reported will be forwarded as Process Error System Messages

3.14 If a Process designated by the SCT as Critical Fails, Subsystem Integrity will publish a Go System Event Code set to 0.

3.15 All changes to the SCT will be recorded in the SDC.

4 Computer Integrity

Computer Integrity monitors each computer and relays the health of the computer to System Integrity.

4.1 Only one copy of Computer Integrity will execute on a computer regardless of the number of subsystems executing on that computer.

4.2 Computer Integrity will provide the following performance data periodically.

4.2.1 Average percent CPU Utilized over the period

4.2.2 Average percent memory available over the period

4.2.3 Network throughput during the period per unit time for each network to which the Computer is attached.

4.2.4 Network interrupts received during the period per unit time for each network to which the computer is attached.

4.2.5 Number of Network Errors since Go for each network to which the computer is attached.

4.2.6 Average disk utilization during the period.

4.2.7 Number of disk accesses

4.2.8 Number of Disk Errors

5 System Configuration Table

The System Configuration Table provides a logical and physical map of the Set. Within a Test set, the data is constrained to the computers and subsystems in the Test Set. *A system-wide SCT is maintained for Set Master operations*. Figure 3 provides an illustration of SCT data and relationships within and external to the SCT.

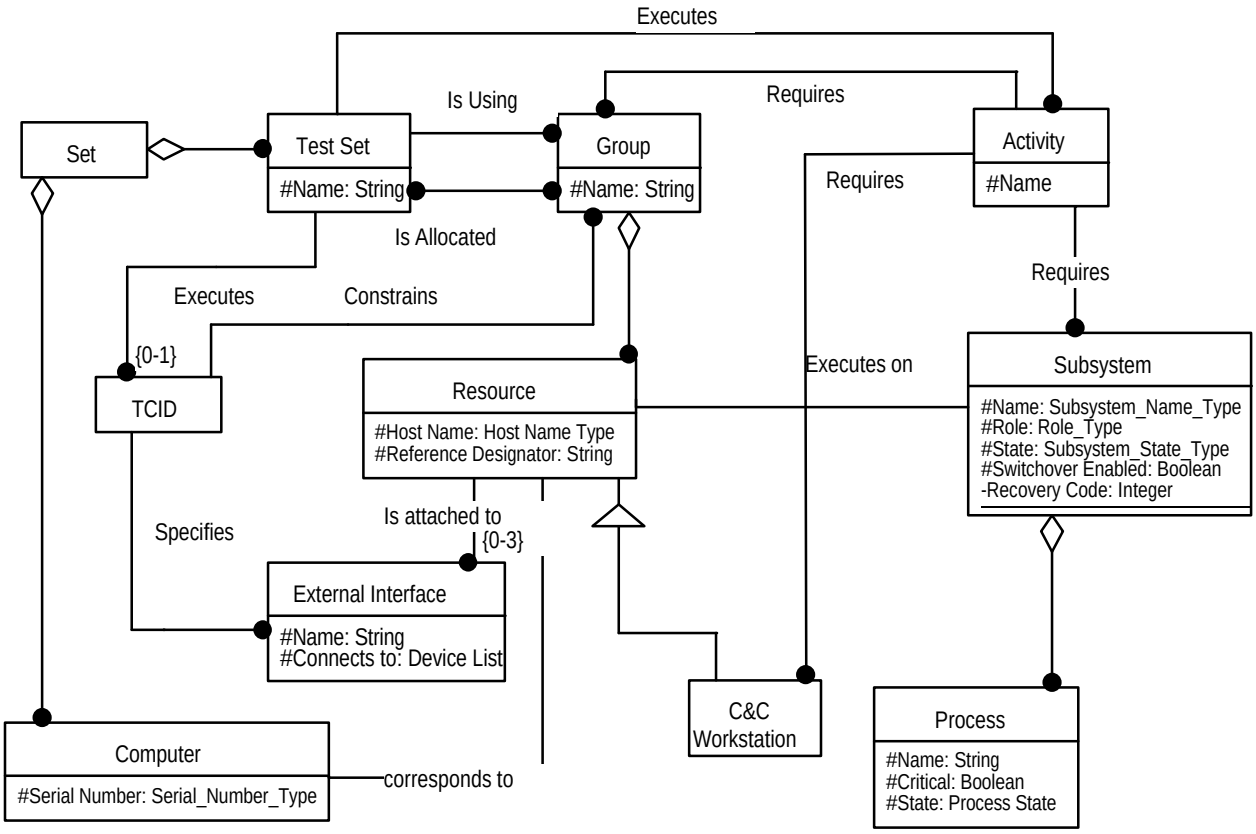


Figure 3 - SCT Class Structure

Bold items are defined in a different 5.x requirement

5.1 The file-based initialization data for the System Configuration Table will be independent of a particular TCID and/or SCID except in the case where the SCT file format varies across the set of SCIDs to be loaded.

5.2 The Redundancy Management CSC will provide an API to allow other applications to retrieve SCT data

5.3 The System Configuration Table will make the following data available for each **Set**:

- 5.3.1 The Name of the Set
- 5.3.2 The **Test Sets** that make up the Set
- 5.3.3 The logical **Resources** that make up the Set.

5.4 The SCT will make the following data available for each logical **Resource**:

- 5.4.1 Host Name/Alias (i.e., lcc_gw_gse1)
- 5.4.2 Reference Designator (i.e., 130A1)
- 5.4.3 Attached **External Interfaces**
- 5.4.4 Physical **Computer**

5.5 The SCT will make the following data available for each **Test Set**

- 5.5.1 Name
- 5.5.2 **Group**s allocated to the Test Set
- 5.5.3 **Group**s in use by the Test Set
- 5.5.4 The Hostname of all **Resources** in the Test Set.
- 5.5.5 CCWS **Resources** allocated to the test set
- 5.5.6 CCWS **Resources** in use by the Test Set
- 5.5.7 The **Subsystems** in the Test Set
- 5.5.8 Gateway **Subsystems** defined for the Test Set.

5.6 The SCT will make the following data available for each **External Interface**
5.6.1 Name (i.e., GSE, LDB)
5.7 The SCT will make the following data available for each physical **Computer**
5.7.1 Serial Number
5.8 The SCT will make the following data available for each **Group**:
5.8.1 Name
5.8.2 Type (e.g., Control Group, Front End Zone)
5.8.3 The **Resources** assigned to the Group
5.9 The SCT will make the following data available for each **Subsystem**:
5.9.1 Name (i.e., GS1A, GS1S)
5.9.2 Role (i.e., Active, Standby, Hot Spare)
5.9.3 State (e.g., In Configuration, Platform Initialized, SCID Initialized, Loaded, Communicating, Go, In ORT)
5.9.4 **Resource** on which the Subsystem is executing
5.10 The SCT will provide an API to allow the a modification to the Resource on which a Subsystem executes.
5.11 SCT changes will be recorded in the SDC.

1.2.3 Redundancy Management CSC Performance Requirements

1 *Set Integrity*

– *None*

2 System Integrity

2.1 System Integrity will detect a missing Computer Heartbeat message within 1 cycle of the expected arrival time.
This means that if the HB is scheduled for frequency of 10ms, that SI will detect the missing packet within 10ms of its expected arrival time. With a failure declared at two missed cycles, this allows for a maximum 30ms detected failure time: If the failure occurs immediately following a heartbeat, 10ms elapses before the first HB is not generated, 10 more before the second is not generated, 10 more before SI realizes that two have been missed. In order to not declare false alerts, and allow for network delays, SI will probably be scheduled to look for the HB somewhere late in the cycle.

3 Subsystem Integrity

3.1 On Active CCPs and DDPs, the Health Counter FD shall be issued at the SSR.
3.2 On Standby CCPs and DDPs, the Health Counter FD shall be issued at 1 Hz.
3.3 On CCWSs, the Health Counter FD shall be issued at the DSR.
3.4 On CCPs, DDPs, and CCWSs, Status FDs shall be issued at 1 Hz.

4 Computer Integrity

4.1 All performance data shall be produced at 1 second intervals.

5 System Configuration Table

5.1 Correct System Configuration data will be provided to an API user within TBD ms of the API invocation. The intent of this requirement is to allow distributed maintenance of the SCT if performance constraints permit.
5.2 A change in the SCT will be visible on all processors within 10 ms of the change.

1.2.4 Redundancy Management CSC Interfaces Data Flow Diagrams

This section describes the interfaces to the RM CSC.

- Subsystem CSCIs provide application health data and can read the SCT as required. The provided health data is used in subsequent SCT updates.
- Ideally, RM will use both Data Fusion and Constraint Management to detect failures in the system. Exact use of these capabilities will be examined in the design phase.
- System Integrity uses System Services to deliver system messages that notify operators of subsystem failures.
- Status is visible to operators through System Viewers that can extract data from the SCT.
- The initial values for the SCT are provided through file(s) exported from Microsoft Access.

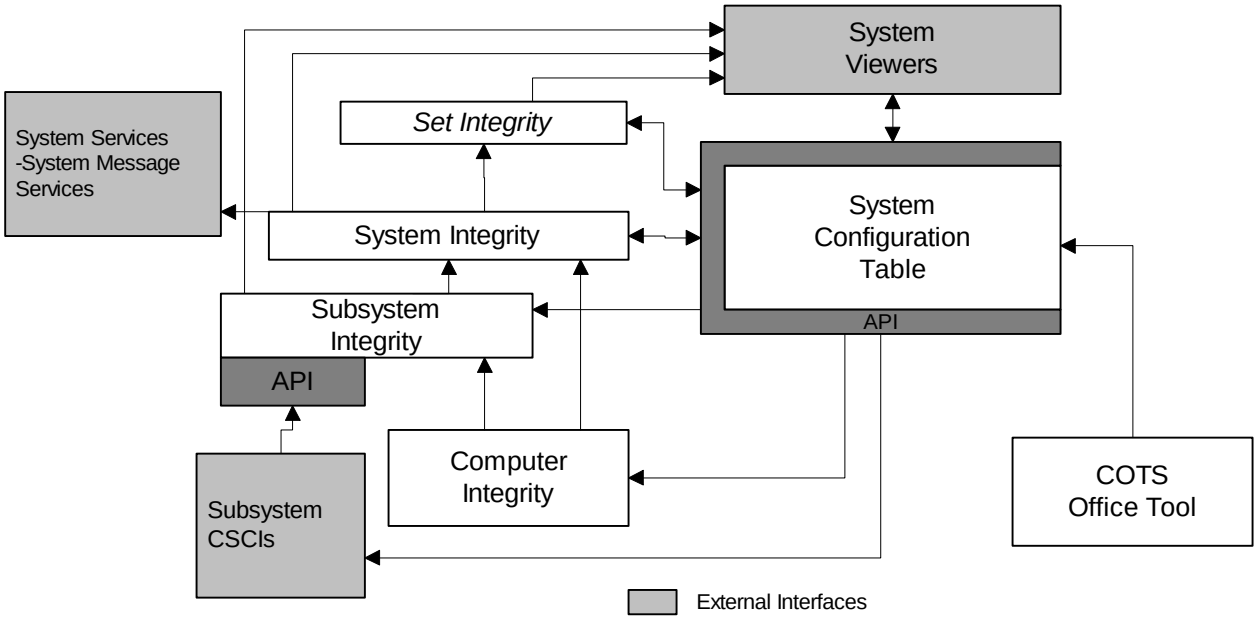


Figure 4 - RM CSC Data Flow

1.3 Redundancy Management CSC Design Specification

As discussed above, the Redundancy Management CSC consists of 5 major parts. Each of these parts are discussed separately below.

1.3.1 Set Integrity

Set Integrity is not provided in Thor.

1.3.2 System Integrity

System Integrity design will be specified in part 2 of the Thor DP3.

1.3.3 Subsystem Integrity

Release Notes: For the first drop of Thor, the APIs are available for use, but are stubbed out.

Subsystem Integrity monitors a single computer. Subsystem Integrity uses both application and hardware level information to produce a summary status for the computer. This summary status is provided to System Integrity as the Health Counter FD. Other Subsystem health data is also provided as FDs. Information used to generate the FDs comes from a number of sources, including system services, system processes, and FDs generated by applications.

1.3.3.1 Subsystem Integrity Detailed Data Flow

1.3.3.2 Subsystem Integrity External Interfaces

1.3.3.2.1 CSC Name Message Formats

TBD

1.3.3.3 Subsytem Integrity Display Formats

This CSC produces no displays

1.3.3.4 Subsystem Integrity Input Formats

All inputs are provided through the API.

1.3.3.5 Subsystem Integrity Recorded Data

1.3.3.6 Subsystem Integrity Printer Formats

This CSC does not print anything.

1.3.3.7 Subsystem Integrity Interprocess Communications

TBD

1.3.3.8 Subsystem Integrity External Interface Calls

Subsystem Integrity provides the following Application Programming Interfaces:

1.3.3.8.1 Class SSIProcess

The SSIProcess Class provides a service to allow Initialization and Termination Services the capability to register a process that they are to create. It also allows deregistration of processes that are to be terminated normally.

1.3.3.8.1.1 Specification

```
class SSIProcess
{
        SSIProcess (string Name, int  PID);
        virtual void Complete ();
        ~SSIProcess ();
};
```

1.3.3.8.1.2 Process

The constructor for the Process class accepts the name of the processing being initiated and the OS Process ID. This information is used to match the initiated process with the process providing a heartbeat.

1.3.3.8.1.2.1 Definition

SSIProcess (string Name, int PID)

Arguments:

- Name is a user supplied string. It should match the name supplied by Process when it creates a Heart Class, or name that will be available through the application health counter.
- PID is the UNIX Process ID for the process being registered.

Return Value: N/A

1.3.3.8.1.3 Complete

The Complete method notifies Subsystem Integrity that the process created by the constructor should no longer be monitored. This interface should only be invoked if there is normal termination of the process. ITS must invoke this method prior to terminating the process so that no race condition is generated for a missed heartbeat.

1.3.3.8.1.3.1 Definition

void Complete();

Arguments: N/A
Return Value: N/A

1.3.3.8.1.4 ~SSIProcess

The destructor cleans up the memory allocated in Subsystem Integrity as well as that in the application program space.

1.3.3.8.1.4.1 Definition

~SSIProcess();

Arguments: N/A
Return Value: N/A

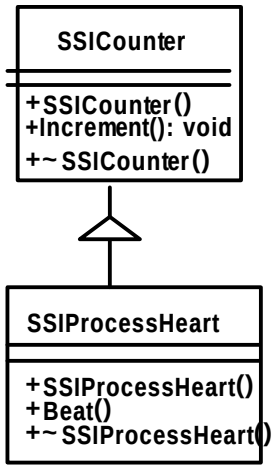
1.3.3.8.1.5 Example

```
SSIProcess *App1Process;                        // declare a pointer to a process

// fork process - returns PID
ApplProcess = new SSIProcess(ApplName, PID);    // register the process

// later, determine that the process should end.
ApplProcess.Complete();                         // Tell SSI not to worry about it
kill Process;                                   // get rid of it
delete ApplProcess;                             // destroy the object instance.
```

1.3.3.8.2 Class SSICounter



The Counter Class provides an interface for data collection. Instances of the counter, declared in application programs, can be monitored by Subsystem Integrity. As specific metrics are identified, it is expected that this class will be further subclassed.

1.3.3.8.2.1 Specification

```
class SSICounter
{
SSICounter (string Name);
virtual void Increment (int by =1);
~SSICounter();
};
```

1.3.3.8.2.2 SSICounter

The constructor creates an instance of the counter visible to both the user and to Subsystem Integrity.

1.3.3.8.2.2.1 Definition

SSICounter (string Name)

Arguments:
Name provides a unique descriptive title under which the statistics are recorded.
Return Value: N/A

1.3.3.8.2.3 Increment

The increment methods increments the counter by the amount specified in “by”. If no value is specified, the count is incremented by 1.

1.3.3.8.2.3.1 Definition

void Increment (int by=1);

Arguments:
by specifies the amount by which the counter should be incremented. If no value is supplied, it will be incremented by 1.
Return Value: N/A

1.3.3.8.2.4 ~SSICounter

This interface destroys the counter object and informs SSI that the data is no longer being supplied.

1.3.3.8.2.4.1 Definition

~SSICounter();

Arguments: N/A
Return Values: N/A

1.3.3.8.2.5 Example

SSICounter DiskAccessCounter("DiskAccesses");

// read from disk
DiskAccessCounter.Increment();
// or if you read and wrote…
DiskAccessCounter.Increment(2)

1.3.3.8.3 Class SSIProcessHeart

The SSIProcessHeart class provides a heartbeat for a process. Each process is expected to either invoke this interface periodically, or to provide an application level Health Counter FD (not the Subsystem Health Counter FD). For latency reasons, using the API is the preferred approach.

1.3.3.8.3.1 Specification:

```
class SSIProcessHeart :: private Counter
{
SSIProcessHeart (string Name, int Period, boolean Periodic);
virtual void Beat ();
~SSIProcessHeart ();
}
```

1.3.3.8.3.2 SSIProcessHeart

This constructor defines to Subsystem Integrity the process to monitor and its expected frequency. It also creates the object in local memory.

1.3.3.8.3.2.1 Definition

SSIProcessHeart (string Name, int Period, boolean Periodic);

Arguments:

- The Name argument uniquely identifies the process to monitor. The name must be unique to the Computer. If the process was registered using the SSIProcess Object, the name must match the name provided at that interface.
- The Period specifies the period of the process in milliseconds. For acyclic processes, this is the maximum amount of time the process will sleep between cycles. MAXINT is considered to be infinite. Note that a process with a MAXINT period cannot be checked for failure directly by SI.
- The Periodic boolean is true if the process is cyclic and false if acyclic. Cyclic processes are monitored for incrementing heartbeats. An incrementing heartbeat must increase by exactly 1 for each period. More or less than 1 is considered a failure. Acyclic processes are monitored for increasing heartbeats. The heartbeat must increase by at least one in the specified period. Increases of more than one are not considered failures. No increase is considered a failure.

Return Value: N/A

1.3.3.8.3.3 Beat

This procedure increments the counter. It should be invoked on every cycle of the process. If not invoked as frequently as specified in the constructor, the process will be considered failed.

1.3.3.8.3.3.1 Definiiton

void Beat()

Arguments: N/A
Return Value: N/A

1.3.3.8.3.4 ~SSIProcessHeart

The destructor destroys the object and notifies SSI that no further hearbeats will be supplied by the process. Note that if the process has not been unregistered by through the SSIProcess.Complete interface, this will result in nofication of a failed process.

1.3.3.8.3.4.1 Definition

~ProcessHeart()
Arguments: N/A
Return Value: N/A

1.3.3.8.3.5 Example

```
ProcessHeart ApplHeart("ApplClient",  TRUE, 1000);          //Cyclic 1Hz process
// or
ProcessHeart Appl2Heart(" ApplClient2",  FALSE, 30000);    //Acyclic process, will wake up at least
                                                           //30 seconds

ApplHeart.Beat();
```

1.3.3.8.4 Class SSISoftwareError

The Software Error class provides a mechanism for an application to report an error to the local copy of Subsystem Integrity. One instance of the class must be created in order to report errors. Subclasses can be used as desired by the application to group error handling. All relevant application identification data is available to SSI through the registration process, and does not need to be supplied through these interfaces.

1.3.3.8.4.1 Specification:

```
enum ErrorType {DATA, EXECUTION, INTERFACE};
enum ErrorSeverity {INFORMATION, WARNING, ERROR, FATAL_ERROR};

class SSISoftwareError
{
SSISoftwareError();
virtual void Report (string ErrorName, ErrorType EType, ErrorSeverity Severity, string
ErrorData);
~SSISoftwareError();
}
```

1.3.3.8.4.2 SSISoftwareError

The constructor creates an instance of the Error object that can then be used to report errors. Multiple instances of Software Error can be created. The constructor also retrieves from the SCT any necessary Process Identification information that will be needed when errors are recorded.

1.3.3.8.4.2.1 Definition

SSISoftwareError()

Arguments: N/A
Return Values: N/A

1.3.3.8.4.3 Report

The Report method is used to notify Subsystem Integrity that an error actually occurred.

1.3.3.8.4.3.1 Definition

void Report (string ErrorName, ErrorType EType, ErrorSeverity Severity, string ErrorData);

Arguments

- ErrorName: This specifies the name to be used to track the error.
- EType: Classifies the error
- Severity: Aids in System Integrity determination of the appropriate response. Specific reactions will be defined on a case by case basis, but general use is that Information Errors are ignored by SI. Warning Errors will be ignored unless some threshold is received, and then used only if it appears the warnings are localized. Exceeding a defined Error rate will result in switchover if available. Fatal Errors will trigger immediate switchover in critical processes.
- ErrorData: Any other data to be recorded that may aid in determining the cause of the error.

Return Value: N/A

1.3.3.8.4.4 ~SSISoftwareError

The destructor should only be invoked as part of process termination.

1.3.3.8.4.4.1 Definition

~SSISoftwareError()

Arguments: N/A
Return Value: N/A

1.3.3.8.4.5 Example

```
SSISoftwareError AllErrors;          // created at program start
…
AllErrors.Report (“FileNotFound”, DATA, ERROR);
…
AllErrors.Report(“InvalidCommand”,INTERFACE,FATAL_ERROR):
```

1.3.3.9 Subsystem Integrity Table Formats

This CSC uses no tables produced by an external source.

1.3.4 Computer Integrity

Computer Integrity design will be specified in part 2 of the Thor DP3.

1.3.5 System Configuration Table

The master copy of the System Configuration Table resides on the Master CCP. *Synchronized copies are maintained on all other computers.*

On each computer, the SCT is initially built from pre-stored files, *then updated based on the changes that have been recorded at the Master CCP.* The SCT is maintained in shared memory, and an API is

provided to allow the SCT users to get access to the information. The system configuration can be viewed as a tree – the Set contains Test Sets, each Test Set is a group of Subsystems, each Subsystem is a collection of processes. The structure of the SCT, and the design of the API reflect this organization. Iterators are provided to allow looping through the sets to extract information, shortcut "My" objects are also defined at each level to allow quick access to local configuration data.

Initial creation of the SCT is through Microsoft Access tables. These tables allow much of the configuration to be defined prior to the start of the test, then updates as the system is configured.

1.3.5.1 System Configuration Table Detailed Data Flow

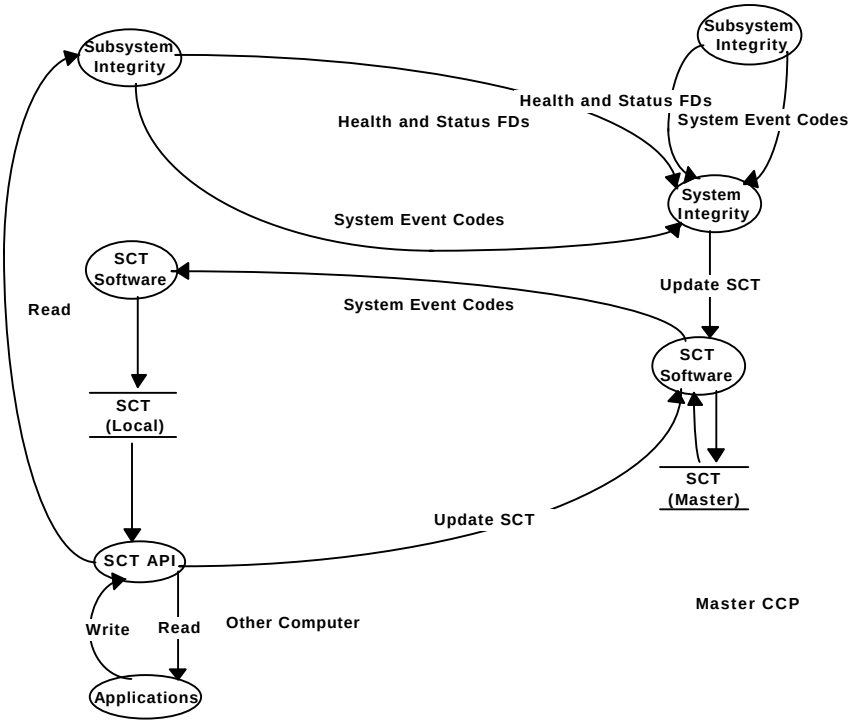


Figure 5 - System Configuration Table Data Flow

When an application at a computer requests a modification to the SCT through the provided API, *the update request is sent to the master copy of the SCT located on the Master CCP. There, the update is made as appropriate, and the other SCTs are updated through System Event Codes.* The updated data is then available to the application through the Read APIs. *Subystem Integrity updates are handled in much the same way. Subsystem Integrity reports health and Status through both System Event Codes and Health and Status FDs. Based on this input, System Integrity evaluates the health of the subsystem and updates the master copy of the SCT. These updates are relayed to all copies of the SCT through System Event Codes.*

1.3.5.2 System Configuration Table External Interfaces

1.3.5.2.1 CSC Name Message Formats

TBD

1.3.5.3 System Configuration Table Display Formats

Microsoft Access Forms are used to generate the SCT data files. The generic Access Table views will be available for the 1/20 drops. Formatted displays will be provided in part 2 of the DP3.

1.3.5.4 System Configuration Table Input Formats

Input formats for the Subsystem Integrity CSC are defined by the FD Formats and the API.

1.3.5.5 System Configuration Table Recorded Data

This information will be provided at drop 2.

1.3.5.6 System Configuration Table Printer Formats

This CSC does not print anything.

1.3.5.7 System Configuration Table Interprocess Communications

This information will be provided at drop 2.

1.3.5.8 System Configuration Table External Interface

This CSC provides an interface for reading data form the SCT as well as writing information into the SCT. The interface is comprised of multiple objects. As shown in Figure 6, the SCT is a tree of containers that reflects the logical configuration of the system. The Set is a collection of Test Sets, Each Test Set contains a number of Subsystems, Resources, Gateways and Groups. Resources are attached to external systems, and Subsystems are composed of Processes.

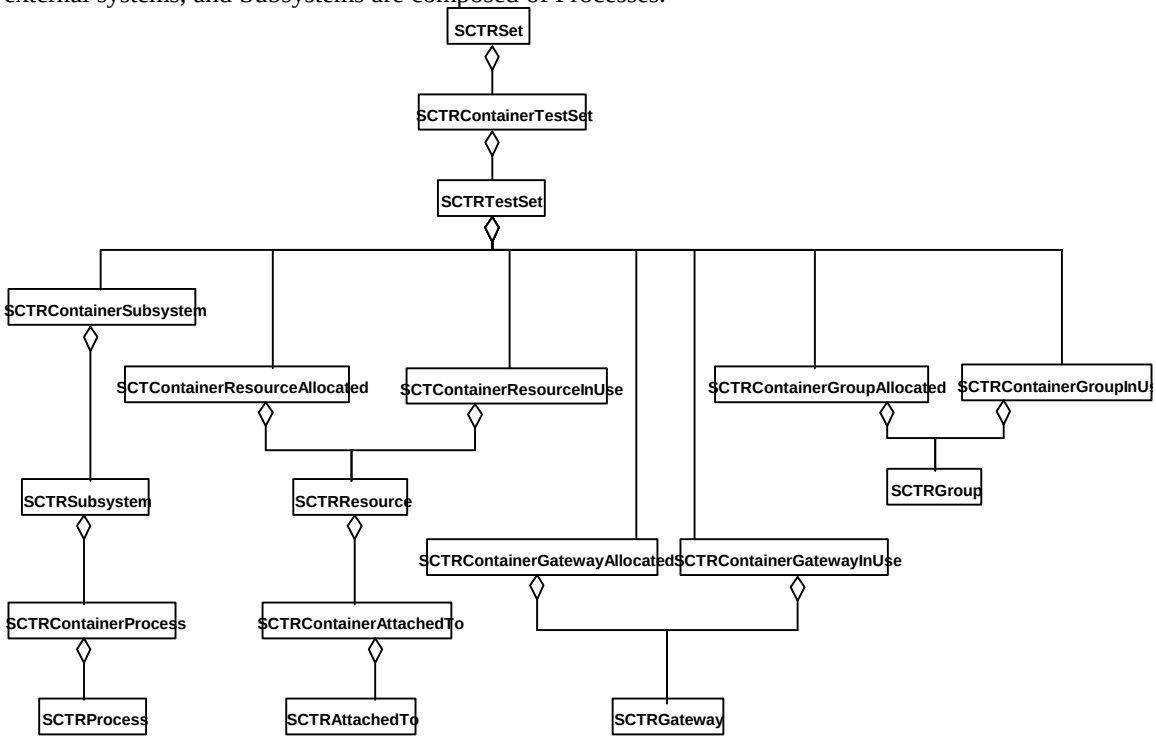


Figure 6 - SCT API Hierarchy

The collections are all derived from the RogueWave class of Ordered (Figure 7). While there is no inherent ordering of the various items, the use of the Ordered class allows searching for objects and stepping through each of the Objects. The SCT Ordered privately inherits the Ordered class methods. Only those methods that manipulate the existing members of the set are exported through the SCT Ordered class. This prohibits deletion of SCT elements through the API. No unique methods are provided in the SCT Ordered subclasses, but the methods are redefined to allow only homogeneous sets.

Figure 8 specifies the methods of each of the classes. In order to be collected, each is derived from the RogueWave RWCollectable class. All values in the classes are available as methods that return the values. The GetContainer methods in each class allow retrieval of the containers as shown in Figure 6. The methods are further specified and explained in the subsections below.

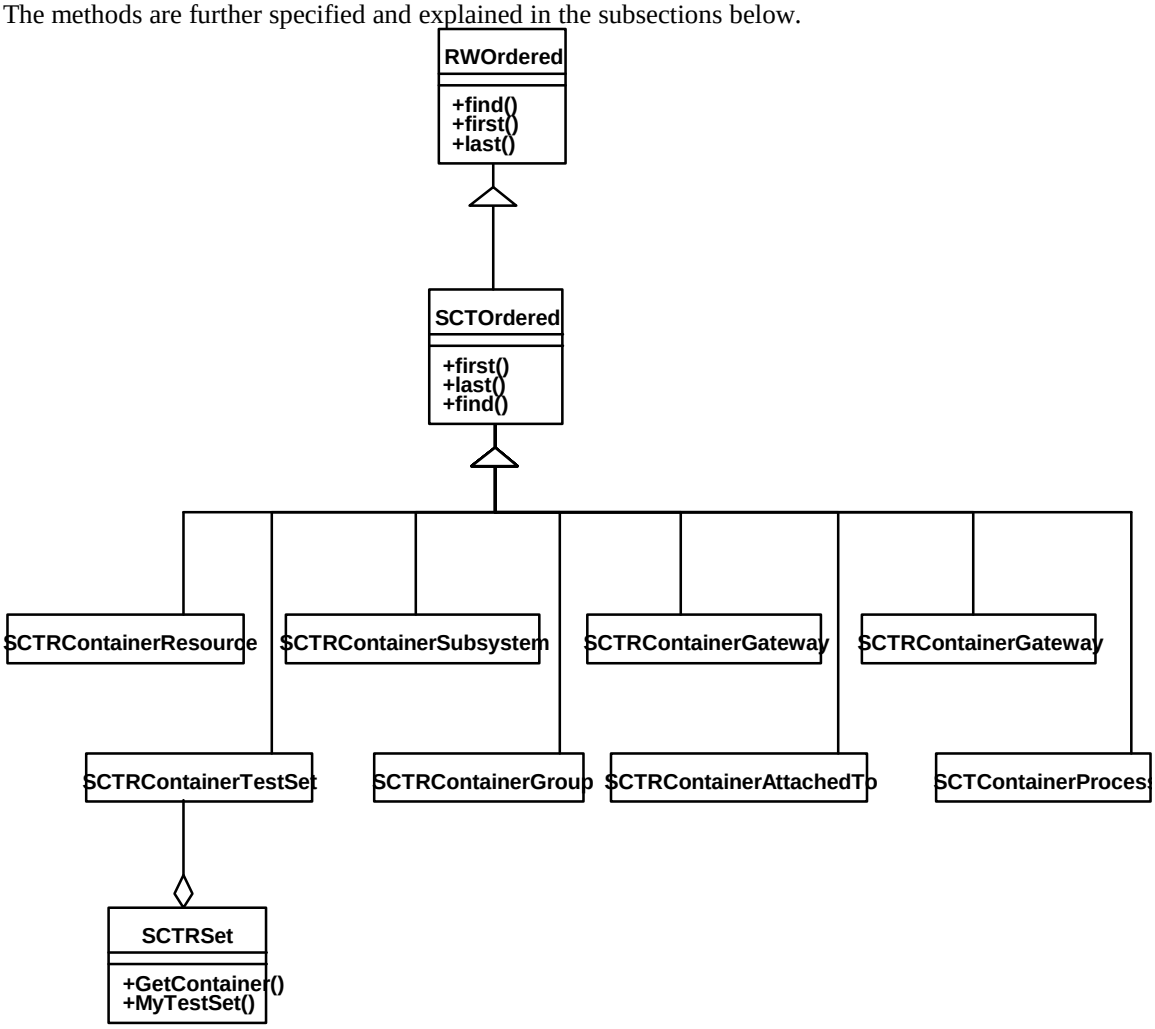


Figure 7 - SCT Container Structure

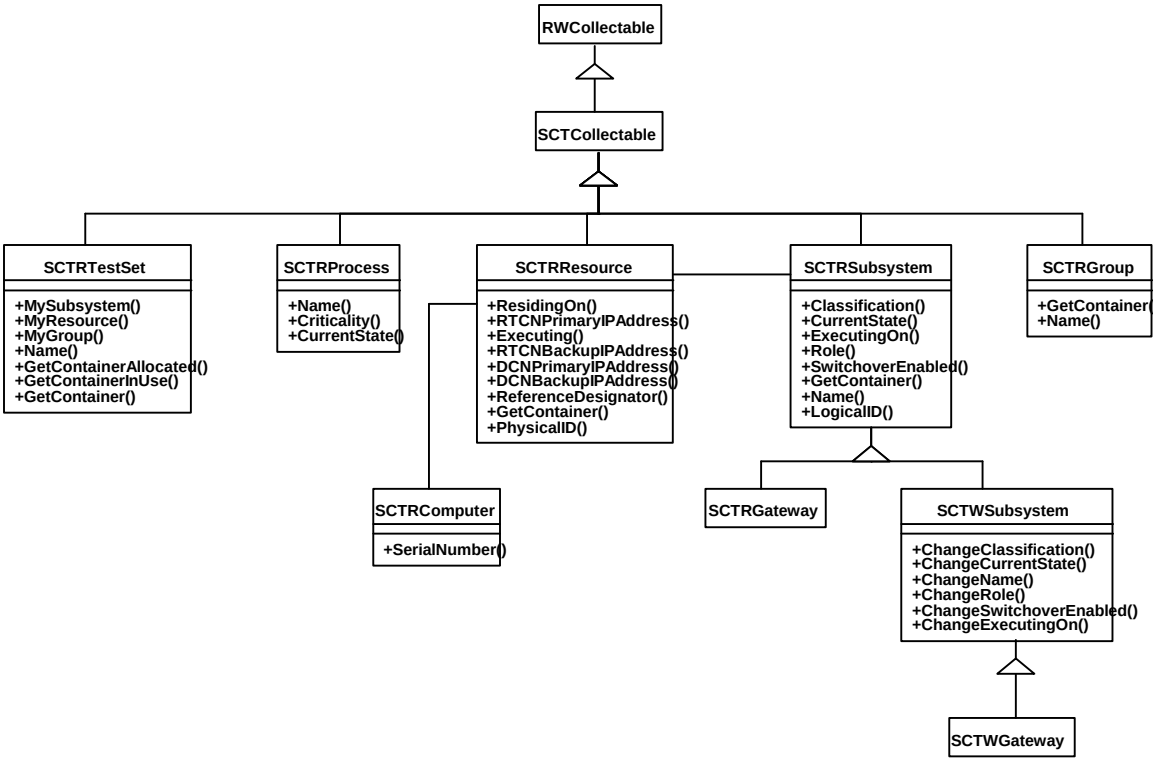


Figure 8 - SCT Object Inheritance

1.3.5.8.1 Class SCTRSet

The SCTRSet Class provides a read only interface for the retrieval of information visible to the Set. The provided methods allow the user to access the Test Set object containing their personal Test Set information as well as provides access to a container having iteration methods to cycle through all of the test sets within the Set. A definition of the available methods as well as examples are shown in the proceeding paragraphs.

1.3.5.8.1.1 Specification

```
class SCTRSet
{
   SCTRSet();
   virtual void GetContainer( SCTRContainerTestSet *pObject );
   virtual void GetContainer( SCTRContainerResource *pObject);
   virtual SCTRTestSet* MyTestSet();
   ~SCTRSet();
};
```

1.3.5.8.1.2 SCTRSet

The SCTRSet constructor will create a Set instance that can provide information in the entire local SCT.

1.3.5.8.1.2.1 Definition

SCTRSet()

Arguments: N/A
Return value: N/A

1.3.5.8.1.3 ~SCTRSet

The ~SCTRSet destructor will destroy a Set instance. This destroys only information created for the using application. The SCT is not affected by this destructor.

1.3.5.8.1.3.1 Definition

~SCTRSet()

Arguments: N/A
Return value: N/A

1.3.5.8.1.4 GetContainer

The GetContainer API provides the Test Set Container or the Resource Container object within a Set.

1.3.5.8.1.4.1 Definition

void GetContainer(SCTRContainerTestSet *pObject)
void GetContainer(SCTRContainerResource *pObject);

Arguments: pObject is a pointer to the type of container object needed.
Return value: N/A

1.3.5.8.1.5 MyTestSet

The GetMy API provides the Test Set object within a Set where the requester resides.

1.3.5.8.1.5.1 Definition

VRTestSet * MyTestSet()

Arguments: N/A
Return value: A pointer to the requester's Test Set object containing their Test Set information.

1.3.5.8.1.6 Examples

```
#include"SCTRAPI.h"
…

SCTRTestSet          *pMyTestSet;        // a Test Set object pointer.
SCTRContainerTestSet *pTestSetContainer; // a Test Set Container object pointer.

SCTRSet *SCT = new SCTRSet;              // allocates a Set object called SCT.

pMyTestSet = SCT.MyTestSet();            // retrieve the requester's Test Set.
SCT.GetContainer( pTestSetContainer );   // retrieve a container with a collection of Test Sets.

delete SCT;                              // remove the allocated Set object.
…
```

1.3.5.8.2 Class SCTRTestSet

The SCTRTestSet provides a read only interface for the retrieval of information visible to the Test Set. Its interface is much the same as the SCTRSet interface with a couple additions. Similar to the SCTRSet interface, this interface provides methods that allow access to personal information such as its Subsystem, its Resource, its Name, etc. Additionally, access to a container is provided having iteration methods allowing the ability to visit all Subsystems, Resources, or Gateways within the Test Set. This interface may be used through the SCTRSet class or if constructed, will provide the Test Set the requester is

currently residing in. A definition of the available methods as well as examples are shown in the proceeding paragraphs.

1.3.5.8.2.1 Specification

```
class SCTRTestSet
{
   SCTRTestSet();
   virtual void GetContainerAllocated( SCTRContainerResource        *pObject );
   virtual void GetContainerAllocated( SCTRContainerGateway         *pObject );
   virtual void GetContainerAllocated( SCTRContainerGroup                  *pObject );
   virtual void GetContainerInUse( SCTRContainerResource            *pObject );
   virtual void GetContainerInUse ( SCTRContainerGateway            *pObject );
   virtual void GetContainerInUse ( SCTRContainerGroup              *pObject );
   virtual void GetContainer( SCTRContainerSubsystem        *pObject );
virtual SCTRSubsystem* MySubsystem();
   virtual SCTRResource* MyResource();
   virtual SCTRGateway* MyGateway();
   virtual SCTRGroup* MyGroup();
   virtual NAMETYPE Name();
   ~SCTRTestSet();
};
```

1.3.5.8.2.2 SCTRTestSet

The SCTRTestSet constructor will create a Test Set instance that only has information pertinent to the requester, i.e., the Test Set they belong to.

1.3.5.8.2.2.1 Definition

SCTRTestSet()

Arguments: N/A
Return value: N/A

1.3.5.8.2.3 ~SCTRTestSet

The ~SCTRTestSet destructor will destroy a Test Set instance.

1.3.5.8.2.3.1 Definition

~SCTRTestSet()

Arguments: N/A
Return value: N/A

1.3.5.8.2.4 GetContainer

The GetContainer APIs provide the requested Container object within a Test Set.

1.3.5.8.2.4.1 Definition

```
   void GetContainerAllocated( SCTRContainerResource     *pObject );
   void GetContainerAllocated( SCTRContainerGateway      *pObject );
   void GetContainerAllocated( SCTRContainerGroup        *pObject );
   void GetContainerInUse( SCTRContainerResource         *pObject );
   void GetContainerInUse ( SCTRContainerGateway         *pObject );
   void GetContainerInUse ( SCTRContainerGroup           *pObject );
```

void GetContainer(SCTRContainerSubsystem *pObject)

Arguments: pObject is a pointer to the type of container object needed.
Return value: N/A

1.3.5.8.2.5 MySubsystem

The MySubsystem API provides a subsystem object containing local subsystem information.

1.3.5.8.2.5.1 Definition

SCTRSubsystem* MySubsystem()

Arguments: N/A
Return value: A pointer to the requester's Subsystem object containing their Subsystem information.

1.3.5.8.2.6 MyResource

The MyResource API provides a resource object containing local resource information.

1.3.5.8.2.6.1 Definition

SCTRResource* MyResource()

Arguments: N/A
Return value: A pointer to the requester's Resource object containing their Resource information.

1.3.5.8.2.7 MyGroup

The MyGroup API provides a group object containing local group information.

1.3.5.8.2.7.1 Definition

SCTRGroup* MyGroup()

Arguments: N/A
Return value: A pointer to the requester's Group object containing their Group information.

1.3.5.8.2.8 Name

The Name API provides the Name of a Test Set object.

1.3.5.8.2.8.1 Definition

NAMETYPE Name()

Arguments: N/A
Return value: The name of this Test Set.

1.3.5.8.2.9 Example(s)

```
#include"SCTRAPI.h"
…

SCTRSubsystem           *pMySubsystem;          // a Subsystem object pointer.
SCTRResource            *pMyResource;           // a Resource object pointer.
SCTRGateway             *pMyGateway;            // a Gateway object pointer.
SCTRGroup               *pMyGroup;              // a Group object pointer.
SCTRContainerSubsystem          *pSubsystemContainer;   // a Subsystem Container object pointer.
SCTRContainerResource  *pResourceContainer;     // a Resource Container object pointer.
```

```
SCTRContainerGateway   *pGatewayContainer;     // a Gateway Container object pointer.
SCTRContainerGroup     *pGrouptContainer;     // a Group Container object pointer.
NAMETYPE               TestSetName;           // the name of this test set.

SCTRTestSet *MyTestSet = new SCTRTestSet;     // allocates a Test Set object called SCT that contains
                                              // the requester's Test Set information..

TestSetName     = MyTestSet.Name();           // retrieve the requester's Test Set name.
PMySubsystem    = MyTestSet.MySubsystem();    // retrieve the requester's Subsystem.
pMyResource     = MyTestSet.MyResource();     // retrieve the requester's Resource.
pMyGateway      = MyTestSet.MyGateway();      // retrieve the requester's Gateway.
pMyGroup        = MyTestSet.MyGroup();        // retrieve the requester's Group.

MyTestSet.GetContainer( pSubsystemContainer );    // retrieve a container with a collection of
Subsystems
                                              // known by the requester's Test Set.
MyTestSet.GetContainerInUse( pResourceContainer );      // retrieve a container with the collection of
                                                        // Resources currently in use by the Test Set.
MyTestSet.GetContainerInUse( pGatewayContainer );       // retrieve a container with a collection of
                                                        // Gateways in use by the requester's Test
                                                        // Set.
MyTestSet.GetContainerAllocated( pGroupContainer );     // retrieve a container with a collection of
                                                        // Groups  allocated to the requester's Test
                                                        // Set.

delete MyTestSet;                             // remove the allocated Test Set object.
…
```

1.3.5.8.3 SCTRGroup

The SCTRGroup provides a read only interface for the retrieval of information visible to the Group. Note that constructor(s) and a destructor are not provided as an interface to this class. It is designed to be utilized through the other classes. A definition of the available methods as well as examples are shown in the proceeding paragraphs.

1.3.5.8.3.1 Specification

```
class SCTGroup
{
   virtual void GetContainer( SCTRContainerResource  *pObject );
   virtual NAMETYPE Name();
};
```

1.3.5.8.3.2 GetContainer

The GetContainer API provides the Resource Container object within a Group. The items in the container are only those items which belong to the group requesting the container.

1.3.5.8.3.2.1 Definition

```
void      GetContainer( SCTRContainerResource *pObject )
```

Arguments: pObject is a pointer to the type of container object needed.
Return value: N/A

1.3.5.8.3.3 Name

The Name API provides the Name of a Group object.

1.3.5.8.3.3.1 Definition

NAMETYPE Name()

Arguments: N/A
Return value: The name of this Subsystem.

1.3.5.8.3.4 Example(s)

```
#include"SCTRAPI.h"
…

SCTRContainerResource  *pResourceContainer;      // a Process Container object pointer.
NAMETYPE               GroupName;                // the name of this test set.

SCTRTestSet *SCT = new SCTRTestSet;                    // allocates a Test Set object called SCT that
                                                       // contains the requester's Test Set
                                                       // information..

pMyGroup        = SCT.MyGroup();                       // retrieve the requester's Group.

(SCT.MyGroup())->GetContainer( pResourceContainer );   // retrieve a container with a collection of
processes
                                                       // known by the requester's Subsystem.
GroupName       = (SCT.MyGroup())->Name();             // retrieve the requester's subsystem name.

delete MyTestSet;                                      // remove the allocated Test Set object.
…
```

1.3.5.8.4 Class SCTRComputer

The SCTRComputer provides an interface to a computer's physical attributes. Note that constructor(s) and a destructor are not provided as an interface to this class. It is designed to be utilized through the other classes. A definition of the available methods as well as examples are shown in the proceeding paragraphs.

1.3.5.8.4.1 Specification

```
class SCTRComputer
{
   virtual SERIALNUMTYPE SerialNumber();
};
```

1.3.5.8.4.2 SerialNumber

The SerialNumber API provides the serial number of the physical computer which a resource is residing on.

1.3.5.8.4.2.1 Definition

SERIALNUMTYPE SerialNumber()

Arguments: N/A
Return value: A value of SERIALNUMTYPE will be returned containing the serial number of this computer object.

1.3.5.8.4.3 Example(s)

The following example assumes that a Test Set object has been created (see the SCRTestSet class examples) and is called SCT.

```
#include"SCTRAPI.h"
…

SERIALNUMTYPE        serialNum;

serialNum = ((SCT.MyResource())->ResidingOn())->SerialNumber();  // retrieve the requestor's computer
                                                                 // serial number.

…
```

1.3.5.8.5 SCTRResource

The SCTRResource provides a read only interface for the retrieval of information visible to a Resource within a Test Set. Note that constructor(s) and a destructor are not provided as an interface to this class. It is designed to be utilized through the other classes. A definition of the available methods as well as examples are shown in the proceeding paragraphs.

1.3.5.8.5.1 Specification

```
class SCTResource
{
   virtual void GetContainer( SCTRContainerAttachedTo *pObject );
   SCTRSubsystem* Executing();
   HOSTNAMETYPE HostName();
   IPTYPE RTCNPrimaryIPAddress();
   IPTYPE RTCNBackupIPAddress();
   IPTYPE DCNPrimaryIPAddress();
   IPTYPE DCNBackupIPAddress();
   REFDESTYPE ReferenceDesignator();
   SCTRComputer* ResidingOn();
   IDTYPE PhysicalID();
};
```

1.3.5.8.5.2 GetContainer

The GetContainer API provides the AttachedTo Container object within a Subsystem providing the ability to access the various connections a resource may have (LDB, GSE, etc).

1.3.5.8.5.2.1 Definition

```
void      GetContainer( SCTRContainerAttachedTo *pObject )
```

Arguments: pObject is a pointer to the type of container object needed.
Return value: N/A

1.3.5.8.5.3 Executing

The Executing API provides the Subsystem executing on this particular Resource.

1.3.5.8.5.3.1 Definition

```
SCTRSubsystem* Executing( )
```

Arguments: N/A
Return value: Returns a pointer to the Subsystem that this Resource is currently executing.

1.3.5.8.5.4 HostName

The HostName API provides the Host Name of a Resource object.

1.3.5.8.5.4.1 Definition

HOSTNAMETYPE HostName()

Arguments: N/A
Return value: Returns the name of the host where this Resource resides.

1.3.5.8.5.5 RTCNPrimaryIPAddress

The RTCNPrimaryIPAddress API provides the IP Address on the Primary RTCN of a Resource object.

1.3.5.8.5.5.1 Definition

IPTYPE RTCNPrimaryIPAddress()

Arguments: N/A
Return value: The IP address of the requested network for this Resource is returned.

1.3.5.8.5.6 RTCNBackupIPAddress

The RTCNBackupIPAddress API provides the IP Address on the Backup RTCN of a Resource object.

1.3.5.8.5.6.1 Definition

IPTYPE RTCNBackupIPAddress()

Arguments: N/A
Return value: The IP address of the requested network for this Resource is returned.

1.3.5.8.5.7 DCNPrimaryIPAddress

The DCNPrimaryIPAddress API provides the IP Address on the Primary DCN of a Resource object.

1.3.5.8.5.7.1 Definition

IPTYPE DCNPrimaryIPAddress()

Arguments: N/A
Return value: The IP address of the requested network for this Resource is returned.

1.3.5.8.5.8 DCNBackupIPAddress

The DCNBackupIPAddress API provides the IP Address on the Backup DCN of a Resource object.

1.3.5.8.5.8.1 Definition

IPTYPE DCNBackupIPAddress()

Arguments: N/A
Return value: The IP address of the requested network for this Resource is returned.

1.3.5.8.5.9 ReferenceDesignator

The ReferenceDesignator API provides the Reference Designator of a Resource object.

1.3.5.8.5.9.1 Definition

REFDESTYPE ReferenceDesignator()

Arguments: N/A
Return value: The reference designator for this Resource is returned.

1.3.5.8.5.10 ResidingOn

The ResidingOn API provides a Computer object on which a Resource object resides.

1.3.5.8.5.10.1 Definition

SCTRComputer* ResidingOn()

Arguments: N/A
Return value: A Computer object pointer where this Resource resides is returned.

1.3.5.8.5.11 Physical ID

The ID API provides the Physical ID of a Resource object.

1.3.5.8.5.11.1 Definition

IDTYPE PhysicalID()

Arguments: N/A
Return value: The ID of this Resource.

1.3.5.8.5.12 Example(s)

```
#include" SCTRAPI.h"
…

SCTRComputer                *pMyComputer;          // my computer object pointer.
SCTRContainerAttachedTo     *pConnections;         // an AttachedTo Container object pointer
for my
                                                   // resource.
HOSTNAMETYPE                HostName;              // the host name of my resource.
IPTYPE                      PrimaryRTCN;           // my resources ip for the primary rtcn.
IPTYPE                      BackupRTCN;            // my resources ip for the backup  rtcn.
IPTYPE                      PrimaryDCN;            // my resources ip for the primary dcn.
IPTYPE                      BackupDCN;             // my resources ip for the backup  dcn.
REFDESTYPE                  MyRefDes;              // my resources reference designator.
IDTYPE                          ResourceId;            // my resources id.

SCTRTestSet *SCT = new SCTRTestSet;                // allocates a Test Set object called SCT that
                                                   // contains the requester's Test Set
                                                   // information..

pMyComputer     = (SCT.MyResource())->ResidingOn();    // retrieve the requester's computer.

(SCT.MyResource())->GetContainer( pConnections );  // retrieve a container with a collection of
processes
                                                   // known by the requester's Subsystem.
MyRefDes        = (SCT.MySubsystem())->ReferenceDesignator();   // retrieve the requester's ref des.
HostName        = (SCT.MyResource())->HostName();  // retrieve the requester's host name.
ResourceId      = (SCT.MyResource())->PhyscialID();    // retrieve the requester's subsystem ID.
```

```
// retrieves the various network ip address.
PrimaryRTCN     = (SCT.MyResource())->RTCNPrimaryIPAddress();
BackupRTCN      = (SCT.MyResource())->RTCNBackupIPAddress();
PrimaryDCN      = (SCT.MyResource())->DCNPrimaryIPAddress();
BackupDCN       = (SCT.MyResource())->DCNBackupIPAddress();

delete MyTestSet;                                   // remove the allocated Test Set object.
…
```

1.3.5.8.6 Class SCTRSubsystem

The SCTRSubsystem provides a read only interface for the retrieval of information visible to a Subsystem within a Test Set. Note that constructor(s) and a destructor are not provided as an interface to this class. It is designed to be utilized through the other classes. A definition of the available methods as well as examples are shown in the proceeding paragraphs.

1.3.5.8.6.1 Specification

```
class SCTRSubsystem
 {
   virtual void GetContainer( SCTRContainerProcess *pObject );
   virtual CLASSIFICATIONTYPPE Classification();
   virtual CURRENTSTATETYPE CurrentState();
   virtual SCTRResource* ExecutingOn();
   virtual SCTRGroup* MyGroup();
   virtual NAMETYPE Name();
   virtual ROLETYPE Role();
   virtual boolean SwitchoverEnabled();
   virtual IDTYPE LogicalID();
};
```

1.3.5.8.6.2 Classification

The Classification API provides the type of a Subsystem object, i.e., CCP, DDP, CCWS, GATEWAY, etc.

1.3.5.8.6.2.1 Definition

CLASSIFICATIONTYPE Clasification()

Arguments: N/A
Return value: The classification of this Subsystem (CCP, DDP, CCWS, GATEWAY, etc.)

1.3.5.8.6.3 CurrentState

The CurrentState API provides the state of a Subsystem object, i.e., COMMUNICATING, GO, etc.

1.3.5.8.6.3.1 Definition

CURRENTSTATETYPE CurrentState()

Arguments: N/A
Return value: The current Subsystem's current state (COMMUNICATING, GO, etc.)

1.3.5.8.6.4 ExecutingOn

The ExecutingOn API provides the Resource object of where this Subsystem is executing.

1.3.5.8.6.4.1 Definition

SCTRResource* ExecutingOn()

Arguments: N/A
Return value: A pointer to the requester's Resource object containing its Resource information.

1.3.5.8.6.5 GetContainer

The GetContainer API provides the Process Container object within a Subsystem.

1.3.5.8.6.5.1 Definition

void GetContainer(SCTRContainerProcess *pObject)

Arguments: pObject is a pointer to the type of container object needed.
Return value: N/A

1.3.5.8.6.6 Name

The Name API provides the Name of a Subsystem object.

1.3.5.8.6.6.1 Definition

NAMETYPE Name()

Arguments: N/A
Return value: The name of this Subsystem.

1.3.5.8.6.7 Role

The Role API provides the role of a Subsystem object, i.e., ACTIVE, STANDBY, HOTSPARE

1.3.5.8.6.7.1 Definition

ROLETYPE Role()

Arguments: N/A
Return value: The roll of this Subsystem (ACTIVE, STANDBY, HOTSPARE)

1.3.5.8.6.8 SwitchoverEnabled

The SwitchoverEnabled API provides a mechanism to determine if the current Subsytem is in switchover.

1.3.5.8.6.8.1 Definition

boolean SwitchoverEnabled()

Arguments: N/A
Return value: boolean (TRUE, FALSE)

1.3.5.8.6.9 LogicalID

The ID API provides the logical ID of a Subsystem object.

1.3.5.8.6.9.1 Definition

IDTYPE LogicalID()

Arguments: N/A
Return value: The ID of this Subsystem.

1.3.5.8.6.10 Example(s)

```
#include" SCTRAPI.h"
…

SCTRResource            *pMyResource;          // a Resource object pointer.
SCTRContainerProcess    *pProcessContainer;    // a Process Container object pointer.
NAMETYPE                SubsystemName;         // the name of this test set.
CURRENTSTATETYPE        SubsystemState;        // the current state of this subsystem.
CLASSIFICATIONTYPE      SubsystemType;         // the classification of this subsystem.
IDTYPE                  SubsystemId;           // the id of this subsystem.

SCTRTestSet *SCT = new SCTRTestSet;            // allocates a Test Set object called SCT that
                                               // contains the requester's Test Set
                                               // information..

pMyResource     = (SCT.MySubsystem())->ExecutingOn();  // retrieve the requester's Resource.

(SCT.MySubsystem())->GetContainer( pProcessContainer );  // retrieve a container with a collection of
                                               // processes known by the requester's
                                               // Subsystem.
SubsystemState  = (SCT.MySubsystem())->CurrentState();   // retrieve the requester's current state.
SubsystemName   = (SCT.MySubsystem())->Name();           // retrieve the requester's subsystem name.
SubsystemId     = (SCT.MySubsystem())->LogicalID();      // retrieve the requester's subsystem ID.

delete MyTestSet;                              // remove the allocated Test Set object.
…
```

1.3.5.8.7 SCTRProcess

The SCTRprocess provides a read only interface for the retrieval of information visible to a Process within a Subsystem. Note that constructor(s) and a destructor are not provided as an interface to this class. It is designed to be utilized through the other classes. A definition of the available methods as well as examples are shown in the proceeding paragraphs.

1.3.5.8.7.1 Specification

```
class SCTRProcess
{
   virtual boolean Critical();
   virtual CURRENTSTATETYPE CurrentState();
   virtual NAMETYPE Name();
};
```

1.3.5.8.7.2 Critical

The Critical API determines if the Process is critical.

1.3.5.8.7.2.1 Definition

```
boolean  Critical()
```

Arguments: N/A
Return value: A boolean indicating the criticality of the process will be returned i.e., TRUE/Critical FALSE/Not Critical

1.3.5.8.7.3 CurrentState

The CurrentState API provides the state of a Subsystem object, i.e., COMMUNICATING, GO, etc.

1.3.5.8.7.3.1 Definition

CURRENTSTATETYPE CurrentState()

Arguments: N/A
Return value: The current state of the process will be returned, i.e., COMMUNICATING, GO, etc.

1.3.5.8.7.4 Name

The Name API provides the Name of a Process object.

1.3.5.8.7.4.1 Definition

NAMETYPE Name()

Arguments: N/A
Return value: The name of the process will be returned of the specified type.

1.3.5.8.7.5 Example(s)

The following example assumes that the user of the process has already retrieved it from its container owned by its subsystem and has called it pAprocess.

```
#include"SCTRAPI.h"
…

   NAMETYPE              ProcessName;
   boolean               IsCritical;
   CURRENTSTATETYPE      ProcessState;

…

   ProcessName  = pAprocess->Name();
   IsCritical   = pAprocess->Critical();
   ProcessState = pAprocess->CurrentState();
….
```

1.3.5.8.8 SCTWSubsystem

The SCTWSubsystem provides a read/write interface for the retrieval and update of information visible to a Subsystem within a Test Set. Its read interface is derived from the SCTRSubsystem class therefore refer to the SCTRSubsystem interface document regarding its use. Note that constructor(s) and a destructor are not provided as an interface to this class. It is designed to be utilized through the other classes. A definition of the available methods as well as examples are shown in the proceeding paragraphs.

1.3.5.8.8.1 Specification

```
class SCTWSubsystem : public SCTRSubsystem
{
   virtual void ChangeClassification( CLASSIFICATIONTYPE newClassification );
   virtual void ChangeCurrentState( CURRENTSTATETYPE newState );
   virtual void ChangeName( NAMETYPE newName );
   virtual void ChangeRole( ROLETYPE newRole );
   virtual void ChangeSwitchover( boolean value );
   virtual void ChangeID( IDTYPE newId );
   virtual RETURNTYPE Update();
```

};

1.3.5.8.8.2 ChangeClassification

The ChangeClassification API provides the ability to modify the classification of a Subsystem object, i.e., CCP, DDP, CCWS, GATEWAY, etc.

1.3.5.8.8.2.1 Definition

void ChangeClasification(CLASSIFICATIONTYPE newClassification)

Arguments: newClassification is an argument of type CLASSIFICATIONTYPE (CCP, DDP, CCWS, GATEWAY, etc.) where its contents will update the local Subsystem[1].
Return value: N/A

1.3.5.8.8.3 ChangeCurrentState

The ChangeCurrentState API provides the ability to modify the current state of a Subsystem object, i.e., COMMUNICATING, GO, etc.

1.3.5.8.8.3.1 Definition

void ChangeCurrentState(CURRENTSTATETYPE newState)

Arguments: newState is an argument of type CURRENTSTATETYPE (COMMUNICATING, GO, etc.) where its contents will update the local Subsystem[1].
Return value: N/A

1.3.5.8.8.4 ChangeName

The ChangeName API provides the ability to modify the current name of a Subsystem object.

1.3.5.8.8.4.1 Definition

void ChangeName(NAMETYPE newName)

Arguments: newName is an argument of type NAMETYPE where its contents will update the local Subsystem[1].
Return value: N/A

1.3.5.8.8.5 ChangeRole

The ChangeRole API provides the ability to modify the current role of a Subsystem object, i.e., ACTIVE, STANDBY, HOTSPARE.

1.3.5.8.8.5.1 Definition

void ChangeRole(ROLETYPE newRole)

Arguments: newRole is an argument of type ROLETYPE (ACTIVE, STANDBY, HOTSPARE) where its contents will update the local Subsystem[1].
Return value: N/A

1.3.5.8.8.6 ChangeSwitchover

The ChangeSwtichover API provides the ability to change[1] the current switchover state of a Subsystem object.

[1] A local update only modifies the writeable Subsystem object private members. It does not alter the contents of the SCT. It is not until a Subsystem Update that the SCT is updated.

1.3.5.8.8.6.1 Definition

void ChangeSwitchover(boolean value)

Arguments: the new switchover state this subsystem is to change to.
Return value: N/A

1.3.5.8.8.7 ChangeID

The ChangeID API provides the ability to change[1] the current Id of a Subsystem object.

1.3.5.8.8.7.1 Definition

void ChangeID(IDTYPE newId)

Arguments: the new Id this subsystem is to change assume.
Return value: N/A

1.3.5.8.8.8 Update

The Update API provides the ability to update the SCT to the changes made to a Subsystem object.

1.3.5.8.8.8.1 Definition

RETURNTYPE Update()

Arguments: N/A
Return value: A return code of type RETURNTYPE (type definition available through the API) which identifies if the update was successful or not.

1.3.5.8.8.9 Example(s)

```
#include"SCTWAPI.h"
…

NAMETYPE            newName;          // the name of this test set.
CURRENTSTATETYPE    newState;         // the current state of this subsystem.
CLASSIFICATIONTYPE  newClassification; // the classification of this subsystem.
IDTYPE                  newId;
ROLETYPE            newRole;
RETURNTYPE          updateResult;

… // appropriate information is stored in the change variables.

SCTRTestSet *SCT = new SCTRTestSet;          // allocates a Test Set object called SCT that
                                             // contains the requester's Test Set
                                             // information..

(SCT.MySubsystem())->ChangeName( newName );          // change my subsystem name.
(SCT.MySubsystem())->ChangeState( newState );        // change my subsystem state.
(SCT.MySubsystem())->ChangeClassification( newClassification ); // change my subsystem
                                                     // classification.
(SCT.MySubsystem())->ChangeID( newId );              // change my subsystem
id.
(SCT.MySubsystem())->ChangeRole( newRole );          // change my subsystem role.
(SCT.MySubsystem())->ChangeSwitchover( TRUE );       // change my subsystem
switchover
                                                     // state to in-switchover (TRUE).
```

```
UpateResult = (SCT.MySubsystem())->Update();                    // change my subsystem role.

delete MyTestSet;                                               // remove the allocated Test Set object.
…
```

1.3.5.8.9 SCTRGateway

The SCTRGateway provides a read only interface for the retrieval of information visible to the Gateway. This object's interface is derived from the SCTRSubsystem interface with no additions. Refer to the SCTRSybsystem interface for definitions and examples on its use.

1.3.5.8.9.1 Specification

```
class SCTRGateway : public SCTRSubsystem
{
};
```

1.3.5.8.10 SCTWGateway

The SCTWGateway provides a read/write interface for the retrieval and update of information visible to the Gateway. This object's interface is derived from the SCTWSubsystem interface with no additions. Refer to the SCTWSybsystem interface for definitions and examples on its use.

1.3.5.8.10.1 Specification

```
class SCTWGateway : public SCTWSubsystem
{
};
```

1.3.5.9 System Configuration Table File Formats

The SCT is built from 4 files. Each file is comma delimited and provides information that defines the part of the system configuration. The format of each of these files is described below.

1.3.5.9.1 Test Set File

This file specifies the Test Sets that can exist in the systems and the Sets in which these Test Sets exist. This is expected to be used primarily by Set Integrity.
Name: testsets.txt

Format: Comma delimited, all strings are enclosed in double quotes.

Field	Type	Contents
Test Set Name	String	A unique name for a test set. There are no constraints on the contents of the string.

Example:
"OCR1 Set 1"
"OCR1 Set 2"

1.3.5.9.2 Group File

This file specifies the Groups of Resources that must be allocated to a single test set. This infomation is used primarily by Set Integrity to ensure that no group is split across test sets.
Name: groups.txt

Format: Comma delimited, all strings are enclosed in double quotes.

Field	Type	Contents

Group Name	String	The Name of the group. This name is unique
Test Set Name	String	A unique name for a test set. This name will exactly match a name in the Test Set File.

Example:

"VAB Gateways","OCR1 Set 1"
"SME Gateways","OCR1 Set 2"
"HMF Gateways","IDE Set"

1.3.5.9.3 Resource File

This file specifies the parameters for the individual resources in the Set.
Name: resources.txt

Format: Comma delimited, all strings are enclosed in double quotes.

Field	Type	Contents
Host Name	String	The commonly used name for the Resource. This name is unique.
Reference Designator	String	This field specify the physical location of the resource
ID	Int16	Number that uniquely identifies the resource within the set.
Primary RTCN IP Address	String	Defines the IP Address to be used by the Resource when communicating on the Primary RTCN. This field may be null (for gateways)
Backup RTCN IP Address	String	Defines the IP Address to be used by the Resource when communicating on the Backup RTCN. This field may be null (for gateways and all processors prior to installation of the backup RTCN).
Primary DCN IP Address	String	Defines the IP Address to be used by the Resource when communicating on the Primary DCN. This field may be null (for CCWSs)
Backup DCN IP Address	String	Defines the IP Address to be used by the Resource when communicating on the backup DCN. This field may be null (for CCWSs and all processors prior to the installation of the backup DCN)
Group	String	The Group to which this resource belongs

Example:
"IDE_CCP01","1","1A365","123.123.255.255","123.123.255.255","123.123.255.255","123.123.255.255"
,
"IDE_CCP02","2","1A366","233.233.233.233","233.233.233.233","233.233.233.233","233.223.223.223"
,
"IDE_DDP01","4","1A399","133.133.133.133","135.13.153.153","111.111.111.111","133.233.233.133",
"IDE_DDP02","3","1A400","233.233.233.233","111.11.111.111","222.222.222.222","133.133.133.133",
"IDE_GSE01","7","2A120","133.233.133.33","22.222.222.222",,,
"IDE_GSE02","6","2A250","144.244.144.44","33.33.33.33",,,"VAB Gateways"

1.3.5.9.4 Subsystem File

This file specifies the definition of the Subsystem, including test set membership and assignment of resources.
Name: subsyst.txt

Format: Comma delimited, all strings are enclosed in double quotes. Types other than strings are not enclosed in quotes

Field	Type	Contents
ID	Int16	Unique identifier for the subsystem. Used as the CPU ID in the network message header.
Subsystem Name	String	The name of the subsystem being defined. Because this file is used by Set Integrity, a given subsystem name may appear more than once. The combination of Test Set/Subsystem Name is unique within the file.
Role	String	This string will specify either "Primary" or "Secondary"
Switchover Enabled	Integer	When 0, Switchover is disabled. When 1 it is enabled. No other value is valid.
Resource Host Name	String	The Host Name on which the Subsystem is to execute.
Test Set	String	Specifies the Test Set for which this mapping is valid. The Test Set name will exactly match one of the names in the test set file.

Example:

```
1,"CCP1A","Primary",1,"IDE_CCP01","IDE Set"
2,"CCP1S","Standby",1,"IDE_CCP02","IDE Set"
3,"GSE1A","Primary",1,"IDE_GSE01","IDE Set"
4,"GSE1S","Standby",0,"IDE_GSE02","IDE Set"
5,"LDB1A","Primary",0,"IDE_GSE01","IDE Set"
```

This is the last page of the Document